Assignment 7: File I/O

CS3423 - Systems Programming — Rocky Slavin - UTSA

For this assignment, you will use **C**'s I/O functions to create a simple inventory system. The system will store basic information about items and allow the user to create, read, update, and delete them. All information for all items will be stored as **binary records in a *single file***.

This assignment requires only the utilities used so far in the I/O lecture notes. **Do not** use bash, sed, awk, find, grep, Python, or any programming languages or utilities besides the C binary functions used in class. Only binary I/O functions should be used to store data to the filesystem (it is OK to use other functions when prompting the user).

Storing Item Information

All item information will be stored in a single binary as record of the following structure where **`itemName`** is the name of the item, **`simpleName`** is a string with no spaces representing an abbreviated version of `itemName`, **`currentQuantity`** is the current amount of items in stock, **`maxQuantity`** is the maximum amount of items, and **`body`** is a description of the item.

```
1 typedef struct
2 {
3     char itemName[64];
4     char simpleName[16];
5     int currentQuantity;
6     int maxQuantity;
7     char body[128];
8 } Item;
```

The program will store all items using the above struct in a single filed called **`inventory.dat`** within the same directory as the program (do not assume this file exists). All items will be referenced by a zero-index item number. Items will be stored in **`inventory.dat`** in item number order. Note that the item number will be specified by the user when an item is entered and will not necessarily be sequential. If **`inventory.dat`** does not exist, one should be created by the program.

Program Execution

When the program is run, the following should occur.

1. Upon running your program, the user should be presented with the following menu:
   ```
   Enter one of the following actions or press CTRL-D to exit.
   C - create a new item
   R - read an existing item
   U - update an existing item
   D - delete an existing item
   ```

2. The user then enters a one-character action (upper or lowercase), leading to one of the following.
   - `C`: an item file is created
     (a) From the terminal, read the following one at a time corresponding to the data structure described above.
        i. Item number **(zero-indexed integer)**
        ii. Simple name
        iii. Item name **(may contain spaces)**
        iv. Current quantity
        v. Maximum quantity
        vi. Description **(may contain spaces)**
     (b) Using the values entered by the user, write the binary structure to `inventory.dat` in the appropriate position.
     (c) You may assume that the user will enter a valid value for each datum.
     (d) If an item already exists for the item number, print the following error and continue with the program. **You should detect this and respond immediately after reading the item number.**
     `ERROR: item already exists`
   - `R`: read an existing item's information
     (a) Prompt the user for an item number:
     `Enter an item number:`
     (b) Locate the specified item using the item number.
     (c) Print the item information in the following format:
     `Item name:` *`item_name`*
     `Simple name:` *`simple_name`*
     `Item Number:` *`item_number`*
     `Qty:` *`current_quantity/max_quantity`*
     `Description:` *`description`*
     (d) If the item is not found, print the following error and continue with the program.
     `ERROR: item not found`
   - `U`: update an existing item
     (a) Prompt the user for the following one at a time
        i. Item number **(zero-indexed integer)**
        ii. Simple name
        iii. Item name **(may contain spaces)**

iv. Current quantity

v. Maximum quantity

vi. Description **(may contain spaces)**

(b) Locate the specified item using the item number.

(c) Update each of the corresponding fields based on the user input. **If the user input is blank for a particular field (except item number), keep the original value from the file.**

(d) If the item is not found, print the following error and continue with the program. **You should detect this and respond immediately after reading the item number.**
`ERROR: item not found`

- `D`: delete an existing item

  (a) Prompt the user for an item number:
  `Enter an item number:`

  (b) Delete the specified item's data from **`inventory.dat`**

  (c) Print the following message with the item's simple name:
  `simple_name` `was successfully deleted.`

  (d) If the item is not found, print the following error and continue with the program.
  `ERROR: item not found`

- If an invalid character is entered, print the following error and continue with the program.
  `ERROR: invalid option`

3. After an action is completed, display the menu again. This should go on indefinitely until `CTRL-D` or the end of a file is reached.

Locating Data

For the above functionality, **`inventory.dat`** should not be read sequentially to search for items. The location of the item in **`inventory.dat`** should be calculated immediately and directly accessed without performing a search.

Hint: You may assume that the maximum quantity will never be zero. Use this to your advantage.

Assignment Data

Input files for testing can be found in **`/usr/local/courses/rslavin/cs3423/Fall18/assign7`** including an existing .dat file and input for stdin. Copy these to your own assignment's directory.

Compiling Your Program

Your submission will include a makefile so the following command can be used to compile your code.

```
$ make assign7
```

Program Files

Your program should consist of up to three files:

- `assign7.c` - the main file which is compiled **(required)**
- `assign7.h` - an optional header file if necessary
- `makefile` - the makefile to make the `assign7` executable **(required)**

Verifying Your Program

Your program must work with the input provided in `a7Input.txt` and `inventory.dat`. To test it:

1. Place `inventory.dat` in the same directory as your compiled program.
2. Execute your compiled program and **redirect** `a7Input.txt` into it. You should not be copying or typing the contents of `a7Input.txt` into your terminal. Redirection must work.
3. Verify that the output is correct based on the input commands.
4. Execute your program again and use your program's menu to test that the information was correctly written to `inventory.dat`.

Submission

Turn your assignment in via Blackboard. Your zip file, named `LastnameFirstname.zip` should contain only your `makefile`, `assign7.c`, and possibly `assign7.h`. Do not include a .dat file.